A Reverse Proxy Is A Proxy By Any Other Name

By Art Stricek
Version 1.2f
January 10, 2002

You just spent the last few months installing, configuring, testing, and securing your outside HTTP presence. In the corner of your office lies the empty coffee cups, tattered manuals and your battered psyche. What you were hoping to do is sit back for awhile sipping your favorite beverage of your choice, right! Well unfortunately the bad guys are not relaxing, they are still contemplating, the meaning of life, the universe and how to violate your inter sanctum. So is there anything else we can do to muddy the attackers water? What about adding a Reverse Proxy to our defenses.

Overview

This paper will cover the concept of a Reverse Proxy by defining what it is and how it differs from a forward proxy. We will cover the benefits and drawbacks of using this technology as a part of our network infrastructure, along with the security advantages and possible risks. This paper will follow the flow of an HTTP request through the firewall, to the Reverse Proxy, to our backend HTTP servers and then back to the requestor. We will also discuss two different Reverse Proxy infrastructure designs and the pros and cons of each of design. And finally we will cover setting up a working Reverse Proxy using the Apache, showing how the concepts of a Reverse Proxy are implemented. Apache is used in this paper for our Reverse Proxy since there is an implementation available for most platforms and it is free. In the end, assuming you already know how to install Apache, you should be able to implement a working Reverse Proxy on an Apache Web Server.

What is A Reverse Proxy

So what exactly is a Reverse Proxy? First let's review what a forward proxy or proxy is and how it works. A forward proxy acts a gateway for a client's browser, sending HTTP requests on the client's behalf to the Internet. The proxy protects your inside network by hiding the actual client's IP address and using its own instead. When an outside HTTP server receives the request, it sees the requestor's address as originating from the proxy server, not from the actual client.

A Reverse Proxy proxies on behalf of the backend HTTP server not on behalf the outside client's request, hence the term reverse. It is an application proxy for servers using the HTTP protocol. It acts as a gateway to an HTTP server or HTTP server farm by acting as the final IP address for requests from the outside. The firewall works tightly with the Reverse Proxy to help ensure that only the Reverse Proxy can access the HTTP servers hidden behind it. From the outside client's point of view, the Reverse Proxy is the actual HTTP server.

Benefits and Drawbacks of A Reverse Proxy

One of the most immediate benefits of using a Reverse Proxy is that clients now have a single point of access to your HTTP servers. This allows you to add another layer to your defenses that might just help you catch an attack against your HTTP servers. Another benefit is that you have a single point of control over who can access and to which HTTP servers you allow access to. A third benefit is the easy replacement of backend servers or host name changes. With a Reverse Proxy these types of changes will not affect the outside clients, because these types of changes are made in the Reverse Proxy rules or mappings; therefore no more messy waiting for names to be republished to the outside DNS world. The idea of having a single point of access also helps in load balancing and failover either by using a DNS round robin scheme or by appliance hardware or software solutions such as F5 Networks Big IP, Cisco's Content Switch, or Macromedia's ClusterCat. Another benefit of the Reverse Proxy is the ability to assimilate various applications running on different Operating Systems behind a single facade. Another advantage is that hardware costs can be lowered significantly because outside and inside clients can access the same servers for the same HTTP requests. This reduction comes from eliminating the duplication of hardware, as usually there are one or more servers for inside clients and one or more servers for the outside clients. Using a Reverse Proxy allows us to secure our backend databases that may be required to service our HTTP server requests without exposing them to the outside world.

Well there must be some sort of downside to the Reverse Proxy, isn't there? This really sounds too good to be true! Yes there are downsides to using a Reverse Proxy; just like anything else, it is not a perfect solution. First if the Reverse Proxy goes south and you do not have failover installed, your entire HTTP presence is down the drain in one fell swoop, not to mention your sleep. Another drawback is that if that an attacker does compromise Reverse Proxy, the attacker may gain more insight into your HTTP server architecture; or if the HTTP servers it is hiding are inside the firewall, the attacker might be able to compromise your internal network. So it is still important to apply all security fixes to your Reverse Proxy and backend servers as soon as they become available and are fully tested and to install Intrusion Detection software on your servers and network. One more drawback is that there are a lot of translations that have to occur for the Reverse Proxy and the firewall to do its translations, so requests may be fulfilled a little bit slower. Another drawback is that attackers are starting to conceal their attacks using HTTP protocol. With all our HTTP traffic going through the Reverse Proxy, we need to start adding content filtering to our arsenal to help ensure that the HTTP requests are actually not attacks. Finally, if you are using analysis-reporting tools against your HTTP servers, you will lose the ability to report on demographics. This is due to the fact that the logs will show that all HTTP requests were from the Reverse Proxy.

How Does the Reverse Proxy Work

Now let's get into just how a Reverse Proxy works. For this scenario we will be placing our Reverse Proxy on a DMZ while the HTTP server(s) are inside our firewall on their own subnet (*see Figure 1*). The site name is ***www.mysite.com***, which resolves to a static NAT address of 10.0.1.1 and a real IP address of 192.168.0.1. This site and NAT address is our Reverse Proxy. Our server inside the firewall is ***myserver.mysite.com*** and has a NAT address of 10.0.100.1 and a real IP address of 192.168.10.1. We set up firewall rules to only allow 192.168.0.1 to access 192.168.10.1 on port 80 (443?) and all port 80 traffic is to be forwarded to 192.168.0.1. The host file on our Reverse Proxy has an entry for the host name myserver.mysite.com and we associate the static NAT address of 10.0.100.1.

Now let's get into just how a Reverse Proxy works. For this scenario we will be placing our Reverse Proxy on a DMZ while the HTTP server(s) are inside our firewall on their own subnet (*see Figure 1*). The site name is ***www.mysite.com***, which resolves to a static NAT address of 10.0.1.1 and a real IP address of 192.168.0.1. This site and NAT address is our Reverse Proxy. Our server inside the firewall is ***myserver.mysite.com*** and has a NAT address of 10.0.100.1 and a real IP address of 192.168.10.1. We

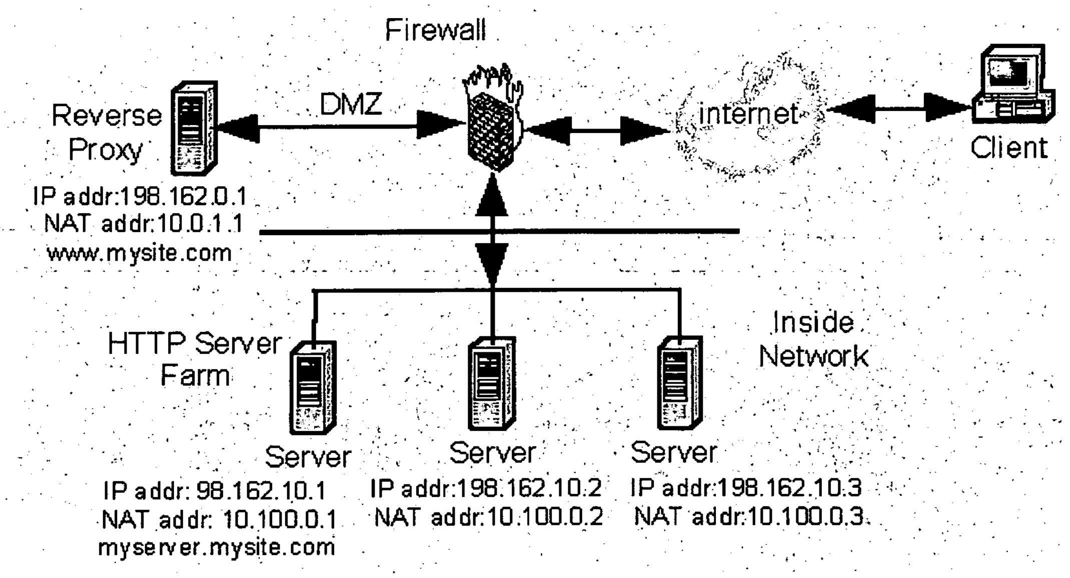

Figure 1

setup firewall rules to only allow 192.168.0.1 to access 192.168.10.1 on ports 80 and 443 and all port 80 and 443 traffic is to be forwarded to 192.168.0.1. The host file on our Reverse Proxy has an entry for the host name myserver.mysite.com and we associate the static NAT address of 10.0.100.1.

In order for the Reverse Proxy to do its magic we need to define prefix mappings. There are two types of prefix mappings – regular and reverse. A regular mapping (*see figure 2*) is used to tell the Reverse Proxy which URL prefix is to be proxied and what the real or final destination URL is.

Source:	myserver.mysite.com
Destination Source:	www.mysite.com

Figure 2

The reverse mapping translates the URL prefix back to the Reverse Proxy's URL (i.e. www.mysite.com). The reverse mapping (*see figure 3*) is what conceals the actual HTTP backend server names and allows it to trap for redirects that might have occurred. This is the most important feature of the Reverse Proxy.

Source:	www.mysite.com
Destination Source:	myserver.mysite.com

Figure 3

Now for the nitty gritty. First a client sends an HTTP GET request to www.mysite.com (10.0.10.1). The firewall receives this request, checks it against its NAT rules, which tell it to forward the request to the Reverse Proxy at IP address 192.168.0.1. The Reverse Proxy receives the request and checks the prefix URL against its mapping, and finds that the request needs to be forwarded to myserver.mysite.com. The Reverse Proxy now forwards the request and again the firewall checks the request against its NAT rules. The firewall sees that the request is from 192.168.0.1 (our Reverse Proxy) and that this address is allowed to access 192.168.100.1 on port 80, so it forwards the request to the inside server. Now myserver.mysite.com processes the request and returns the results back to the requestor, the Reverse Proxy. The Reverse Proxy again checks the request against its reverse mappings and changes all references in the URL and the HTTP header from myserver.mysite.com to www.mysite.com. After completing all translations, the Reverse Proxy now forwards the request back to the client who originally sent the HTTP GET request. From the client side it appears that the request was satisfied by www.mysite.com, but in reality it was satisfied by myserver.mysite.com. As you can see it is pretty much a slight of hand game, which makes it harder for the attacker to gain access to your crown jewels.

Reverse Proxy Infrastructure

There are several considerations to consider in setting up a Reverse Proxy infrastructure. The first and most important is security. Using static NAT'd addresses to reference all servers is one of the most important. If an attacker is performing reconnaissance against your site, using the firewall to translate the NAT addresses to real addresses makes it a lot more difficult to gather the real information about your infrastructure. The firewall rules also play a major role in the Reverse Proxy. Setting up the firewall rules to route HTTP traffic to the Reverse Proxy and only allowing requests to HTTP backend servers to originate from the Reverse Proxy creates a single gateway allowing us to watch all HTTP traffic from this single point.

Another point to keep in mind is that only the hostname of the Reverse Proxy is published to the wild, wild Internet. All other HTTP server hostnames that the Reverse Proxy will be forwarding requests to are stored in the etc/hosts file. Only assign static NAT'd addresses to the hostnames in the host file on the Reverse Proxy. Intrusion Detection needs to be set up on the Reverse Proxy and all backend servers that it is forwarding requests to, along with monitoring all log files. For the proxy server, you can use Apache, Microsoft Proxy, Squid or any other proxy that supports regular and reverse mapping.

There are at least two ways to set up the Reverse Proxy infrastructure. In this paper we will cover two of them. The first we have already covered (*see figure 1*). In this infrastructure design the backend servers are on our internal network behind the firewall and the Reverse Proxy is on the DMZ. The backend HTTP servers need to be on their own subnet to segregate them from the rest of the internal network. If these backend servers are Windows NT or Windows 2000 servers and are using Windows authentication, then a separate Domain must be set up. This Domain has no trust relationships with the inside network. This is to ensure that if they are compromised that the attacker does not gain easy access to our inside network. If inside clients need to access these HTTP servers, then new userid's and cryptic passwords must be assigned to the clients under this Domain. This to prevent an attacker gaining access to our internal network if he is successful in getting these external IDs and cracks the passwords. The advantage of this infrastructure is that external and internal clients can use one set of servers, which reduces hardware costs. No replication of content is required, which reduces bandwidth costs. Administration costs are reduced because there is only one set of servers for both inside and outside, plus the servers can be administered just like any other server on our internal network making the Administrator's task easier. The drawback of this infrastructure is that if the Reverse Proxy is compromised and an attacker gains access to the backend HTTP servers, the attacker could gain access to our internal network. In setting up the individual server operating systems, follow the SANS guidelines for securing each platform.

So now I know you are saying to yourself, why in the world would I ever implement such an infrastructure that might allow an attacker to gain access to my internal network! I know it looks risky at first glance, but lets review the security features of this particular infrastructure. First no one has direct access to our backend servers, only the Reverse Proxy can access them directly. Second, in order to access the backend servers, the http request will have to pass through four intrusion detection points, the firewall, the Reverse Proxy, the firewall and finally the backend server (see section *"How Does the Reverse Proxy Work"*). Add content filtering and you now have five intrusion detection points. Third, Administration work is all done behind the protection of the firewall, so Administrator userids and passwords are not out in the open for an attacker to grab. And last but not least, if the Reverse Proxy is compromised, the attacker can only gain access to servers on that subnet. When combined with the all the benefits of a Reverse Proxy, then this infrastructure becomes a very attractive alternative.

In the second design we place the backend HTTP servers and the Reverse Proxy on the DMZ (*see figure 4*). Once again we NAT all backend HTTP server addresses and do not include them in our external DNS. Instead we use the etc/hosts file for our hostname to IP address lookups. If some or all of these backend servers are using Windows NT or 2000, then we need to set up an external Domain for authentication and follow the SANS guidelines for securing IIS and Windows 2000. Once again the firewall acts as a traffic cop, making sure that only HTTP traffic goes to our Reverse Proxy. The difference is that the Reverse Proxy forwards the requests directly to the backend servers. The advantage of this setup is that our internal network is totally separated from the outside world and if compromised, the attacker can gain no

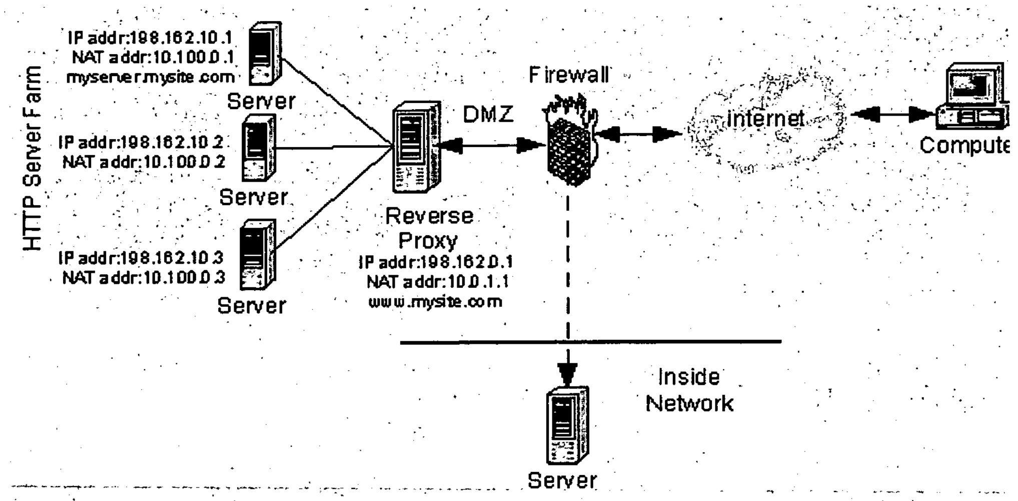

Figure 4

insight to our internal network. We still have a single point of access for all clients, and a single main point to monitor for attacks. The drawback is that now we have to create a replication process to keep our HTTP servers in sync with internal servers. We may also have two sets of servers, servers for inside clients and servers for outside clients. This adds to our costs by requiring more hardware and administrator costs for maintaining, patching and troubleshooting more servers. Administrators most likely will need to physically go to the servers to maintain them, or at best have a restricted toolkit to maintain them.

Setting up the Reverse Proxy

For this paper we will be using the Apache Server as our Reverse Proxy and the infrastructure we defined in figure 1. In order to configure Apache, you will need to be able to switch to the root account. To be able to editing the httpd.conf file and the hosts file. Hopefully no one has the ability to logon directly as root. The httpd.conf file is where all the startup configuration information is stored for Apache. We will not go into all the settings in the configuration file, for that is a story for another rainy day. We are only going to deal with those settings required for setting up and securing the Reverse Proxy. One document on securing Apache you might want to review is the NSA guide "Secure Configuration of the Apache Web Server" located at the following URL http://nsa1.www.conxion.com/support/download.htm. The actual location where the httpd.conf file differs from each distributor of operating vendor, so you may have to issue a find to locate it.

O.K. let's rollup our sleeves and get into the real stuff. The first thing we need to do is tell the firewall what address and port requests are to be forwarded to the Reverse Proxy. We will also need to tell the firewall what servers the Reverse Proxy can access inside our firewall. We are not going to go into how to implement the firewall rules in this paper. Next we need to edit our hosts file and add the hosts that the Reverse Proxy will be forwarding requests to.

Contents of etc/hosts

```
127.0.0.1              localhost                       loopback
#
#  Local subnet addresses
#
192.168.10.1            www.mysite.com                 www
192.168.10.2            www.myothersite.com            myothersite
#
#  NAT address for internal hosts
#
10.0.100.1             myserver.mysite.com             myserver
10.0.100.2             mydomino.mysite.com             mydomino
10.0.100.3             httpappl.mysite.com             httpappl
```

Contents of httpd.conf

If you are using pre-packaged binary of Apache you need to make sure the following modules are uncommented out in order to implement Reverse Proxying in Apache. If you are compiling Apache you will need to make sure you include these modules in your compile. The two most important modules in this list are the rewrite_module and the proxy_module. The rewrite module is what allows us to implement regular mapping and the proxy module is what allows us to implement our reverse mapping. This is not a definitive list of what modules should or should not be included, but it is meant to be a starting point.

```
LoadModule rewrite_module      libexec/mod_rewrite.so
LoadModule proxy_module        libexec/libproxy.so
LoadModule vhost_alias_module  libexec/mod_vhost_alias.so
LoadModule env_module          libexec/mod_env.so
LoadModule define_module       libexec/mod_define.so
LoadModule config_log_module    libexec/mod_log_config.so
LoadModule mime_module          libexec/mod_mime.so
LoadModule negotiation_module   libexec/mod_negotiation.so
LoadModule alias_module         libexec/mod_alias.so
LoadModule access_module        libexec/mod_access.so
LoadModule unique_id_module     libexec/mod_unique_id.so
LoadModule setenvif_module      libexec/mod_setenvif.so

<IfDefine SSL>
LoadModule ssl_module          libexec/libssl.so
</IfDefine>
```

Note: If you add or remove any module in the above section, you must do the same in the module list below.

```
ClearModuleList
AddModule  mod_vhost_alias.c
AddModule  mod_env.c
AddModule  mod_define.c
AddModule  mod_log_config.c
AddModule  mod_mime.c
AddModule  mod_negotiation.c
AddModule  mod_alias.c
AddModule  mod_rewrite.c
AddModule  mod_access.c
AddModule  mod_proxy.c
AddModule  mod_unique_id.c
AddModule  mod_so.c
AddModule  mod_setenvif.c

<IfDefine SSL>
   AddModule mod_ssl.c
</IfDefine>
```

We now need to define which ports our Reverse Proxy will be listening to for normal http and secured http requests.

```
Port 80

<IfDefine SSL>
   Listen 80
   Listen 443
   Listen 7301
</IfDefine>
```

Next we need to tell Apache what our server name is and where our default document root is. Since the proxy will not be serving as an actual web server, we will define a new directory that all our virtual document roots will use.

```
ServerName www.mysite.com

DocumentRoot "/virtual/mysite/tmp"
```

Now we want to set the default services and features to be very restrictive. From this point forward you must specifically allow a particular feature or service.

```
<Directory />
    Options -FollowSymLinks -SymLinksIfOwnerMatch IncludesNoExec
    AllowOverride None
</Directory>
```

The next statement should have the same value you set for the DocumentRoot above

```
<Directory "/virtual/mysite/tmp">

    Options None
```

The following statement control which options the .htaccess files can override.

```
    AllowOverride None
```

The next statement controls who can retrieve from this server

```
    Order allow,deny
    Allow from all

</Directory>

AccessFileName .htaccess

#
# The following lines prevent .htaccess files from being viewed by
# Web clients.  Since htaccess files 0ften contain  authorization
# information, access is disallowed for security reasons.  Also,
# folks tend to use names such as .htpasswd for password files, so
# this will protect those as well. (Note from HP-UX httpd.conf)
#

<Files ~ "^\.ht">
    Order allow,deny
    Deny from all
</Files>
```

Now we need to setup the default logs for Apache. This section defines where log information goes if you do not specify a log file and directory for a Virtual Host

```
ErrorLog /logfiles/default/error.log
CustomLog /logfiles/default/access.log combined
```

In order for Apache to use a single IP address and port for multiple hosts you need to use the NameVirtualHost directive.

If you want to use name-based virtual hosts you need to define at least one IP address (and port number) for them. This allows you to save on IP addresses.

```
NameVirtualHost 198.162.0.1:80
NameVirtualHost 198.162.0.1:443
NameVirtualHost 198.162.0.1:7403
```

Now we need to define our default VirtualHost. This is the host we will publish as our outside presence and it will be our Reverse Proxy.

```
# Reverse Proxy - www.mysite.com

<VirtualHost 198.162.0.1:80>
   DocumentRoot /virtual/tmp
   ServerName www.mysite.com
   ErrorLog /logfiles/mysite/unsecured_error.log
   CustomLog /logfiles/mysite/unsecured_access.log combined
```

You must add the following statements and set them to on for Reverse Proxying. In order for us to define a regular map to our Reverse Proxy we need to be able to rewrite the incoming URL and HTTP headers. We do this by using Apache's Rewrite module. The RewriteLogLevel can be set to a value from 1 to 9. The Rewrite log shows us a trace of Apache matching the URL to each RewriteRule and which rule it matched against. It also shows exactly how the URL has been rewritten (*see figure 7*). The level of detail is determined by the RewriteLogLevel, with a value of one being the least amount of detail and a value of 9 being verbose.

Do not set ProxyRequests on . Setting this to "on" when using Apache as a Reverse Proxy permits the abuse of the reverse proxy as a forward proxy

```
ProxyRequests off
```

#The next three lines are often not required, as they will only fix up HTTP Redirects not content based
URLs.

```
RewriteEngine on
RewriteLog /logfiles/mysite/unsecured_rewrite.log
RewriteLogLevel 1
```

Turn off all services and features, since they are not needed for Reverse Proxying.

```
<Directory />
```

```
        Options None
        AllowOverride None
    </Directory>
```

The following statements are what comprise our regular mappings for our reverse proxying the main site www.mysite.com. In Apache we accomplish this feat by using the rewrite directive. The rewrite command uses regular expressions to rewrite the client's request and then send it to the real backend server. We only need to use the root on for our sites because the virtual has already recognized the site (www) and domain (mysite.com) and has stripped them off. The syntax of the rewrite rule is:

RewriteRule *pattern substitution* [*flags*]

The way that the RewriteRule works is that it matches the incoming URL to the *pattern* – if it matches then the *substitution* is made. The flags we will be using are defined in figure 5. Some basic regular expression symbol meanings are defined below in figure 6.

Flag	Abbreviation	Purpose
redirect	R	Forces a redirect
proxy	P	Forces the request to be proxied
	NC	Not case sensitive
last	L	Last rule; ignore any other rules that follow. Process what is in the *substitution*

Figure 5 - RewriteRule Flags

Symbol	Meaning
^	Beginning of line
$	End of line
()	Keep value between parens as a variable, starting with $1, and pass it to the substitution
.*	Return all remaining characters

Figure 6 – Regular Expression Symbols

One rule to try to adhere to when writing the RewriteRule is **Keep It Simple**. Regular expressions can be used to compose very complex matches, but the more complicated they are, the more chance of accidentally creating an opportunity for an attacker. With the basics out of the way here are the regular mappings of our Reverse Proxy.

The following rewrite rule is used to rewrite a request intended for an HTTP application. This application has no web pages, but does transmit information via the HTTP protocol.

```
# URL is www.mysite.com/httpappl2.

RewriteRule ^/(httpappl2)/(.*)$  http://myserver.mysite.com/$1/$2 [NC,P]
```

The next two rewrites are needed to allow the Reverse Proxy to handle Domino webs. We need to redirect notes because domino expects to find all references relative from the default site. Since the default site here is not a domino server, we need to redirect it to another site (www3) so that all icons and graphics are accessible.

```
RewriteRule ^/(mynotes)/(.*)$     http://www3.mysite.com/$1/$2    [NC,R]
RewriteRule ^/(mynotes)/(.*)/$    http://www3.mysite.com/$1/$2    [NC,R]
```

All remaining requests are to be passed to myserver.mysite.com

```
RewriteRule ^/(.*)$               http://myserver.mysite.com/$1/  [NC,P]
```

The following statements are what comprise our reverse mappings for our reverse proxying. Apache has the directive ProxyPassReverse to handle the reverse mappings. The syntax for ProxyPassReverse is:

ProxyPassReverse *path virtualhost*

Where path is the path to the backend server and virtualhost is the URL of the virtual you want returned to the client making the HTTP request.

```
   ProxyPassReverse /            http://myserver.mysite.com/httpappl2/

</VirtualHost>
```

The </VirtualHost> ends the host definition.

This next virtual is used by the redirect in our previous virtual host www.mysite.com. Its sole purpose is to handle Reverse Proxying of Domino web sites.

```
# www3.mysite.com this is a virtual host to handle Lotus Notes Domino

<VirtualHost 198.162.0.1:80>
   DocumentRoot /virtual/tmp
   ServerName www3.mysite.com
   ErrorLog /logfiles/www3/unsecured_error.log
   CustomLog /logfiles/www3/unsecured_access.log combined

   RewriteEngine on
   RewriteLog /logfiles/www3/unsecured_rewrite.log
   RewriteLogLevel 1

   <Directory />
      Options None
      AllowOverride None
   </Directory>

   RewriteRule ^/(domino)/(.*)$  http://mydomino.mysite.com/$1/$2 [NC,P]
   RewriteRule ^/(.*)$           http://mydomino.mysite.com/$1    [NC,P]

   ProxyRequests on
   ProxyPassReverse / http://mydomino.mysite.com/
   ProxyPassReverse / http://mydomino.mysite/
   ProxyPassReverse / http://notesappl.mysite.com/

</VirtualHost>
```

This next virtual is an example of Reverse Proxying a non-web HTTP application that used port 7301.

```
<VirtualHost 198.162.0.1:7301>
   DocumentRoot /virtual/tmp
   ServerName www.myothersite.com
   ErrorLog /logfiles/myothersite /unsecured_error.log
   CustomLog /logfiles/myothersite /unsecured_access.log combined

   RewriteEngine on
   RewriteLog /logfiles/myothersite/unsecured_rewrite.log
   RewriteLogLevel 1

   <Directory />
      Options None
      AllowOverride None
   </Directory>

   RewriteRule ^/(.*)$       http://myotherserver.mysite.com/$1   [NC,P]

   ProxyRequests on
   ProxyPassReverse /        http:// myotherserver.mysite.com/

</VirtualHost>
```

And lastly here is an example of Reverse Proxying a secured socket (SSL) site.

```
<VirtualHost 198.162.0.1:443>
   DocumentRoot /virtual/tmp
   ServerName www.mysite.com
   ErrorLog /logfiles/mysite /secured_error.log
   CustomLog /logfiles/mysite /secured_access.log combined

   RewriteEngine on
   RewriteLog /logfiles/mysite/secured_rewrite.log
   RewriteLogLevel 1

   <Directory />
      Options None
      AllowOverride None
   </Directory>
```

```
    RewriteRule ^/(.*)$        http://myserver.mysite.com/$1   [NC,P]

    ProxyRequests on
    ProxyPassReverse /         http:// myotherserver.mysite.com/

</VirtualHost>
```

Testing Your Reverse Proxy

So now that we have a Reverse Proxy configured, how can we test it. Here are some of the steps in testing your server.

- Change your RewriteLogLevel in your httpd.conf file to a value of 9. This will show you exactly how the Rewrite module is rewriting your URL.
- Open a browser and request a web page from a web server that is being front-ended by the Reverse Proxy. Review of the logs on the Reverse Proxy should show that the request was sent to the proper remapped backend server.
- Do the same if you are accessing an HTTP application server through the Reverse Proxy by having the application issue an HTTP GET to a backend server)
- Review your Rewrite log file to make sure your rewrite rule is remapping your URL the way you want (*see Figure 7*).

```
10.0.0.20 - - [07/Jan/2002:13:16:28 -0500] [www.mysite.com/sid#4006b2a0][rid#400a8118/initial]
(2) init rewrite engine with requested uri /
10.0.0.20 - - [07/Jan/2002:13:16:28 -0500] [www.mysite.com/sid#4006b2a0][rid#400a8118/initial]
(3) applying pattern '^/(httpappl2)/(.*)$' to uri '/'
10.0.0.20 - - [07/Jan/2002:13:16:28 -0500] [www.mysite.com/sid#4006b2a0][rid#400a8118/initial]
(3) applying pattern '^/notes$' to uri '/'
10.0.0.20 - - [07/Jan/2002:13:16:28 -0500] [www.mysite.com/sid#4006b2a0][rid#400a8118/initial]
(3) applying pattern '^/notes/$' to uri '/'
10.0.0.20 - - [07/Jan/2002:13:16:28 -0500] [www.mysite.com/sid#4006b2a0][rid#400a8118/initial]
(3) applying pattern '^/(httpappl2)$' to uri '/'
10.0.0.20 - - [07/Jan/2002:13:16:28 -0500] [www.mysite.com/sid#4006b2a0][rid#400a8118/initial]
(3) applying pattern '^/(.*)$' to uri '/'
10.0.0.20 - - [07/Jan/2002:13:16:28 -0500] [www.mysite.com/sid#4006b2a0][rid#400a8118/initial]
(2) rewrite / -> http://myserver.mysite.com/
10.0.0.20 - - [07/Jan/2002:13:16:28 -0500] [www.mysite.com/sid#4006b2a0][rid#400a8118/initial]
(2) forcing proxy-throughput with http://myserver.mysite.com/
10.0.0.20 - - [07/Jan/2002:13:16:28 -0500] [www.mysite.com/sid#4006b2a0][rid#400a8118/initial]
(1) go-ahead with proxy request proxy:http://myserver.mysite.com/ [OK]
```

Figure 7 - RewriteLog Example

- Now look at the backend server's log and you should see an HTTP request coming from the Reverse Proxy's address.
- Review the address line on the browser. It should have the prefix URL of the Reverse Proxy and not the actual backend servers URL.
- If you are accessing an HTML page, then view the source for the returned HTML page. You should not see any URL references to the backend server.
- Send an HTTP GET request via netcat to several server addresses that are not being front-ended by the Reverse Proxy. You should not be able to see the backend server's address, in fact you should receive a 404 – Not found error.

Troubleshooting

- If you received a 404 – Not Found or a timeout, then check your rewrite rule and make sure RewriteRule changed the URL correctly.
- Check your proxy logs to make sure that it received the request and in deed it really did forward it to the backend server
- Review your firewall logs to make sure it is not being denied by the firewall.
- If the rewrite rule is correct and there are no denials in the firewall then review your static NAT rules to make sure they are defined correctly.

Conclusion

Just to recap here are some of the do's, don'ts, pros and cons to keep in mind when setting up a Reverse Proxy.

Do

- Place backend servers that the Reverse Proxy will access on a segregated subnet and or Domain
- Set Firewall rules to only allow outside access to backend servers from the Reverse Proxy
- All HTTP content references should use non-qualified addresses (no domain prefixes)
- Use aliases and static NAT addresses to reference the Reverse Proxy
- If authentication is required, assign different users and cryptic passwords to access backend servers that are being accessed by the Reverse Proxy
- Install intrusion detection and current patches on the Reverse Proxy and all backend servers
- Use etc/hosts file instead of DNS to reference backend servers
- Review all log files on a regular schedule

Don't

- Allow trust relationships between inside servers and backend servers that are accessed by the Reverse Proxy
- Publish backend servers' addresses and host names in DNS
- Reference the actual IP address of backend servers – use etc/hosts file with static NAT addresses
- **NEVER** use the same outside network user's and passwords as the ones assigned for inside network access
- Place any other web content or web servers on the same server as the Reverse Proxy – make sure that the server is used for Reverse Proxying only

Pro

- Single point of control, monitoring and logging
- Clients have a single point of access
- Reduced hardware, software and maintenance costs
- Ability to have single client interface while assimilating different web platforms and OS's behind it
- Ability to hide backend HTTP servers
- Relatively inexpensive to implement

Con

- In the event of a failure, all connectivity is gone unless failover is implemented
- Attacker may gain insight into your network infrastructure if compromised
- May gain access to internal network if not setup properly
- Firewall and proxy translations may slow down response

References:

Print References

- Laurie, Ben, Laurie, Peter, "Apache The Definitive Guide" Second Edition, O'Reilly & Associates, 1999, (2, 62-169,170-178)
- Wainwright, Peter, "Professional Apache", Wrox Press Ltd., 1999,(150, 163-182, 286-305)
- Loutonen, Ari, 'Web Proxy Servers", Prentice Hall, 1998 (325-343)

Web Site References

- Engelschall, Ralph S., "Load Balancing Your Web Site", WebTechniques, 1998,01-17-2001 http://www.webtechniques.com/archives/1998/05/engelschall/
- Netscape Proxy Manual, Reverse Proxy, 01-17-2001 http://developer.netscape.com/docs/manuals/proxy/adminnt/revpxy.htm
- Engelschall, Ralph. S., "URL Rewriting Guide", December 1997, 01-17-2001 http://www.engelschall.com/pw/apache/rewriteguide/
- Apache Module mod_proxy, 01-17-2001 http://httpd.apache.org/docs/mod/mod_proxy.html